MET O 11 TECHNICAL NOTE NO 73

Matrix multiplication by diagonals

R DIXON

1. Introduction

Vector and array processors will admit of new algorithmic techniques. The Appendix to this note consists of a paper by N.K. Madsen, G.H.Rodrigue, and J.I.Karush in which they establish three "Fortran Theorems" which they claim have relevance to these new processors.

It is not the purpose of this note to assess the validity of their claims as this is a matter for a programmer. The purpose of this note is to relate their Fortran ideas to standard matrix algebra. It seems desirable as a general principle that, where possible, these new types of algorithm should be related to standard matrix algebra in order to facilitate the study of further possibilities and applications.

N.K.MADSEN, G.H.RODRIGUE, and J.I.KARUSH:- "Matrix multiplication by diagonals on a vector/parallel processor" - Information Processing Letters, 5, 2, June 1976, pp 41-45.

N.B. This paper has not been published. Permission to quote from it must be obtained from the Assistant Director of the above Meteorological Office branch.

2. The Matrix Algebra

What is usually meant by the matrix multiplication of two matrices A and B is the scalar product, denoted in texts by AB or A•B. Specifically, for a 3 x 3 example if

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad \text{and} \quad B=\begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \qquad (1)$$

then if C= A•B we have

$$C=\begin{pmatrix} (a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}) & (a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}) & (a_{11}b_{13}+a_{12}b_{23}+a_{13}b_{33}) \\ (a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}) & (a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}) & (a_{21}b_{13}+a_{22}b_{23}+a_{23}b_{33}) \\ (a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31}) & (a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32}) & (a_{31}b_{13}+a_{32}b_{23}+a_{33}b_{33}) \end{pmatrix}$$

the well-known result. (2)

Another less well-known product of two matrices is the Hadamard product. In terms of A and B as given by (1) it is defined by

$$A \square B= \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{pmatrix} \qquad (3)$$

that is, each element of A is multiplied by the corresponding element of B. As an aside from the theory, it is noticeable immediately that this is a convenient type of product if the elements of each matrix are strung out as vectors in a computer store.

Suppose now that instead of forming $A \square B$ we form $A \square \tilde{B}$, the Hadamard product of A with the transpose of B. The result is

$$A \square \tilde{B}= \begin{pmatrix} a_{11}b_{11} & a_{12}b_{21} & a_{13}b_{31} \\ a_{21}b_{12} & a_{22}b_{22} & a_{23}b_{32} \\ a_{31}b_{13} & a_{32}b_{23} & a_{33}b_{33} \end{pmatrix} \qquad (4)$$

It is now apparent by inspection of (2) and (4) that the elements on the main diagonal of A•B are the sums of the elements of the rows of $A \square \tilde{B}$. This means that by introducing the column vector of ones

$$\underline{1}= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \qquad (5)$$

the main diagonal of A•B can be written as a vector $\underline{c}_o$ as

$$\underline{c}_o = (A \square \tilde{B}) \bullet \underline{1} \tag{6}$$

The other diagonals can also be written in terms of Hadamard products by inventing a little special notation. For example, let

$${}_2A \square {}^2\tilde{B}$$ mean - take the bottom two rows of A and the top two rows of $\tilde{B}$ and form the Hadamard product

and let $\underline{c}_{-2}$, $\underline{c}_{-1}$, $\underline{c}_o$, $\underline{c}_1$, $\underline{c}_2$ be the diagonals of A•B as vectors starting at the bottom left corner of (2), then

$$\begin{aligned}
\underline{c}_{-2} &= ({}_1A \square {}^1\tilde{B}) \bullet \underline{1} \\
\underline{c}_{-1} &= ({}_2A \square {}^2\tilde{B}) \bullet \underline{1} \\
\underline{c}_o &= (A \square \tilde{B}) \bullet \underline{1} \\
\underline{c}_1 &= ({}^2A \square {}_2\tilde{B}) \bullet \underline{1} \\
\underline{c}_2 &= ({}^1A \square {}_1\tilde{B}) \bullet \underline{1}
\end{aligned} \tag{7}$$

3. Hadamard Products

The algebra of the Hadamard products is not easy to come by, being scattered through the literature over about seventy years. Most textbooks give only a passing reference to it, if at all. It has been variously called the Hadamard product, the Schur product, and the Ring product, and it has been denoted by $*$, $\circ$, $\square$, $\boxdot$, $\odot$, and $\times$. Lately, opinion has been settling down in favour of the name Hadamard product and either of the two symbols $\circ$ or $*$. I have used $\square$ as a personal preference simply because $\circ$ is much used to denote composition in algebra and $*$ is usually used to denote an undefined product in vector analysis. Some elementary results which may be useful are

(a) If A and B are of unit rank then they can be expressed in terms of vectors as $A = \underline{u}\tilde{\underline{v}}$ and $B = \underline{w}\tilde{\underline{x}}$. In this case

$$A \square B = (\underline{u}\tilde{\underline{v}}) \square (\underline{w}\tilde{\underline{x}}) = (\underline{u} \square \underline{w})(\tilde{\underline{v}} \square \tilde{\underline{x}}) \tag{8}$$

(b) With $\underline{1}$ defined as in (5) above $J = \underline{1}\tilde{\underline{1}}$ is a matrix of ones. It plays the role of identity matrix in Hadamard products. Thus

$$A \square J = J \square A = A \tag{9}$$

(c) Hadamard multiplication is commutative (unlike the usual scalar product multiplication). Thus

$$A \square B = B \square A \tag{10}$$

(d) It is distributive

$$(A + B) \square C = A \square C + B \square C \tag{11}$$

(e) $$\mathrm{Tr}(A \bullet B) = \tilde{\underline{1}} \bullet (A \square \tilde{B}) \bullet \underline{1}$$

(f) If D is a diagonal matrix then

$$D\bullet(A\square B) = (D\bullet A)\square B = A\square(D\bullet B) \tag{12}$$

$$(A\square B)\bullet D = A\square(B\bullet D) = (A\bullet D)\square B \tag{13}$$

$$D\bullet(A\square B)\bullet D = (D\bullet A\bullet D)\square B = A\square(D\bullet B\bullet D) = (D\bullet A)\square(B\bullet D) = (A\bullet D)\square(D\bullet B) \tag{14}$$

(g) If A has the form $A = \underset{\sim}{u}\underset{\sim}{v}$, let D_u and D_v be diagonal matrices formed $\underline{u}$ and $\underline{v}$. Then

$$A\square B = (\underset{\sim}{u}\underset{\sim}{v})\square B = (D_u\bullet J\bullet D_v)\square B = D_u\bullet(J\square B)\bullet D_v = D_u\bullet B\bullet D_v \tag{15}$$

There is a good deal more at a more esoteric level but the matter can rest here pending the development of interest and utilization.

R Dixon
Met O 11

October 1976

APPENDIX

Volume 5, number 2 INFORMATION PROCESSING LETTERS June 1976

MATRIX MULTIPLICATION BY DIAGONALS ON A VECTOR/PARALLEL PROCESSOR *

Niel K. MADSEN, Garry H. RODRIGUE and Jack I. KARUSH
Lawrence Livermore Laboratory, University of California, Lovermore, California, USA

Received 1 December 1975, revised version received 29 March 1976

Analysis of algorithms, numerical mathematics, matric multiplication, parallel processing

1. Introduction

The advent of vector and parallel computers has forced the reexamination, reformulation, and rethinking of essentially all of the basic mathematical algorithms. In this paper, we will consider the seemingly straightforward process of matrix multiplication. We will be primarily concerned with this process on a vector processor, the CDC STAR-100.

For large full matrices, matrix multiplication is easily "vectorized" when the matrix is stored by columns in the typical Fortran fashion. However, there are at least two disadvantages to this normal approach. First, it becomes quite inefficient for banded matrices with relatively narrow bandwidths. Second, when a matrix is stored by columns (or rows), the transpose of the matrix is not as readily available for use in a vector form (this is particularly a problem on the STAR-100).

The purpose of this paper is to present a new algorithm for matrix multiplication which: is readily "vectorized", is very efficient for narrow banded matrices, and allows for the transpose to be easily accessible in a vector form.

2. Storing matrices

The key to the success of the new algorithm is the manner in which one stores a given matrix. For years, mathematicians have viewed matrices largely in terms of rows and columns. This is reflected very strongly in the scientific programming languages such as Fortran or Algol, which assume row or column type storage. For vector/parallel processors, other options have certain advantages. We will show that to implement matrix multiplication it can be very convenient (especially in the case of banded matrices) to store the matrices by diagonals instead of by rows or columns. Storage by diagonals for banded matrices is not new; however, the direct use of only these diagonals as the vectors in a multiplication algorithm is new and is the main point of this paper.

To explain the new algorithm, we first establish some convenient notation. For a given $n \times n$ matrix $A = (a_{ij})$, we define the vectors $\boldsymbol{a}_{\pm k}$, $k = 0, 1, \ldots, n-1$ to have mth components as follows: $\boldsymbol{a}_k(m) \equiv a_{m,k+m}$, and $\boldsymbol{a}_{-k}(m) \equiv a_{k+m,m}$, for $m = 1, \ldots, n-k$. More explicitly, the vectors $\boldsymbol{a}_k$ and $\boldsymbol{a}_{-k}$ represent the diagonals of the matrix A, Clearly the length of the vectors $\boldsymbol{a}_{\pm k}$ is $n - k$. We also will need the concept of the vector "offset". Let $\boldsymbol{v} = (v_1, \ldots, v_m)$ be a vector of length m. The notation $\boldsymbol{v}(p;q)$ will denote the vector $\boldsymbol{v}(p;q) = (v_{p+1}, v_{p+2}, \ldots, v_{m-q})$. We say that $\boldsymbol{v}(p;q)$ has been obtained by offsetting the vector $\boldsymbol{v}$ by p from the left (top) and q from the right (bottom). We also will use the shortened notation: $\boldsymbol{v}(p;) \equiv \boldsymbol{v}(p;0)$ and $\boldsymbol{v}(;q) \equiv \boldsymbol{v}(0;q)$. In the remainder of the paper, unless otherwise noted, all vectors will denote diagonals of a matrix.

There are two obvious advantages to storing or structuring a matrix by its diagonals. First, this method is "natural" for banded matrices in the sense that the

* Work performed under the auspices of the U.S. Energy Research and Development Administration under Contract No. W-7405-Eng-48.

matrix is defined and stored in terms of a very *few* vectors which are as *long* as possible. Second, the transpose of A, A^T, is readily available in terms of the *same* vectors $a_{\pm k}$. With a matrix structured or stored in the usual manner (by columns), A^T cannot be easily described in terms of vectors whose elements are stored consecutively in the computer memory. Consecutive storage is absolutely essential on the STAR-100 computer. It is less important on other machines such as the TI-ASC or the Burroughs ILLIAC-IV.

The main disadvantages to storing matrices by diagonals are: that one's training and intuition, and hence, ability to analyze, are all based on row or column storage; the diagonals have different lengths; and, there are more diagonals than there are rows or columns. Therefore, matrix algorithms are more difficult and less intuitive to describe and understand (at least initially).

3. Conventional multiplication algorithms

Given the $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$, the formation of the product $AB = C = (c_{ij})$ in the usual manner, i.e., $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$, requires n^3 scalar (as contrasted to vector) multiplications and about n^3 scalar additions or n^2 inner products of respective row and column vectors of A and B of length n. Since a vector inner product machine instruction exists on the STAR-100, one might be tempted to use it. However, due to the relatively long execution time of the inner product instruction, this method may be discarded immediately as a candidate algorithm (on the STAR-100).

Another technique [1] is the following. Let c_j denote the *column vector* whose elements consist of the jth column of C. Clearly we have that

$$c_j = \sum_{k=1}^{n} b_{kj} a_k , \qquad (1)$$

where a_k is the kth column vector of A. The algorithm given by (1) is a very efficient algorithm on any currently existing vector/parallel processor for full matrices when n is sufficiently large. Since the multiplication of a vector by a scalar is preformed as a vector multiplication, the complete algorithm would require a total of n^2 vector multiplications and $(n-1)^2$ vector additions (all operations being of length n). Now, the time T required for a typical vector operation on the STAR-100 is usually expressed in units of minor cycles as $T = S + n/R$ where S is the vector start-up time, n is the length of the vector operand(s), and R is the number of results generated per minor cycle. The timings for vector addition and multiplication on the STAR-100 are given by $S = 94$ and 156, and $R = 2$ and 1, respectively. Therefore, the algorithm given by (1) would require a minimum of

$$[250n^2 - 188n + 94] + \tfrac{3}{2}n^3 - n^2 + \tfrac{1}{2}n$$

minor cycles to execute. The bracketed quantity is the total time consumed by vector start-ups. This time will be small relative to the total time only if the vector operands are sufficiently long, i.e., n is sufficiently large. There are two cases when short vectors are simply unavoidable when using this algorithm. The first occurs when n is small, i.e., the size of the problem is small. This, of course, is a problem that perplexes all algorithms when implemented on a vector processor such as the STAR-100. The second case occurs when A and B are matrices with narrow bandwidths. It is for this case that the algorithm presented in the next section is designed to be very efficient.

4. Matrix multiplication by diagonals

The basic conceptual difference between the new algorithm in this section and the conventional algorithm is that instead of forming a *column* of C as the basic computation (1), we will form a *diagonal* of C. The algorithm to compute a diagonal $c_{\pm k}$ of C is somewhat tedious to describe mathematically, but has a simple graphic representation as illustrated in fig. 1. To compute c_k, the kth diagonal of C with $k \geq 0$, one simply deletes the *bottom* k rows of A and the top k rows of B^T. Next, the resulting two congruent $n - k \times n$ matrices are multiplied diagonal by diagonal (element by element) to form the $n - k \times n$ matrix D_k. Finally, the diagonals of D_k are added to each other to form the final result vector c_k whose mth component is the sum of the elements in the mth row of D_k. To form c_{-k} for $k \geq 0$, one starts by deleting the *top* k rows of A and the *bottom* k rows of B^T and then proceeds as above.

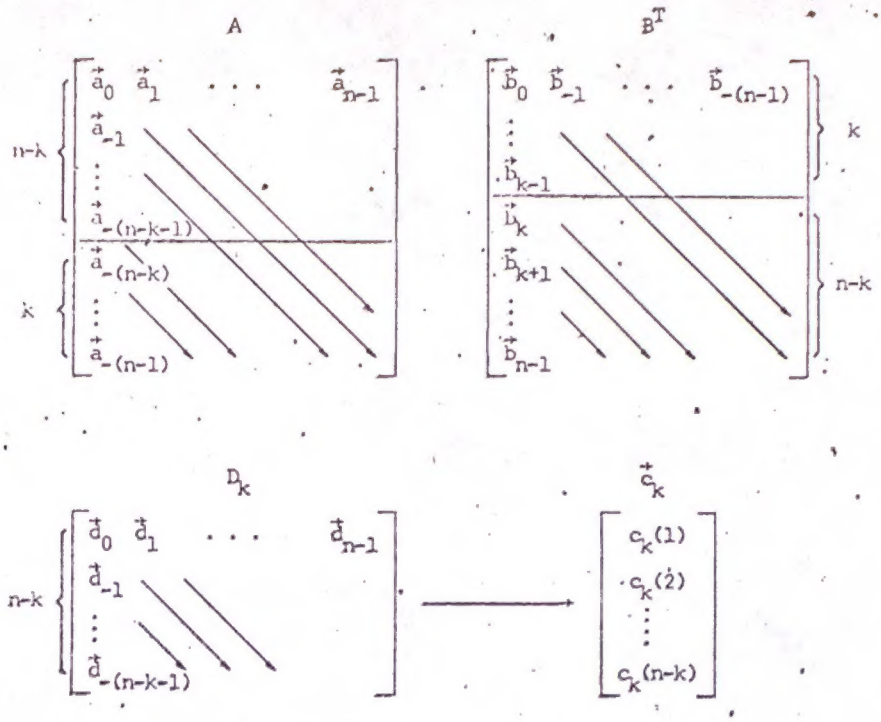

Fig. 1. Graphic representation of matrix multiplication by diagonals.

The following result can be verified in a straightforward, albeit lengthy, computation. The use of fig. 1 is helpful.

Theorem 1. Let $A = (a_{-(n-1)}, ..., a_0, ..., a_{n-1})$ and $B = (b_{-(n-1)}, ..., b_0, ..., b_{n-1})$ be $n \times n$ matrices stored by the indicated diagonals. Then $AB = C = (c_{-(n-1)}, ..., c_0, ..., c_{n-1})$ is given by the following "Fortran-like" algorithm: let $c_k \equiv 0$ for all k, then

i) for $k = 0, 1, ..., n-1$, compute c_k by

$$c_k(i-k;) = c_k(i-k;) + a_{k-i}(;k)b_i$$

for $i = k+1, ..., n-1$,

$$c_k(;i-k) = c_k(;i-k) + a_i b_{k-i}(k;)$$

for $i = k+1, ..., n-1$,

$$c_k = c_k + a_i(;k-i)b_{k-i}(i;)$$

for $i = 0, 1, ..., k$,

ii) For $k = 1, 2, ..., n-1$ compute c_{-k} by

$$c_{-k}(i-k;) = c_{-k}(i-k;) + a_{-i}b_{i-k}(;k)$$

for $i = k+1, ..., n-1$,

$$c_{-k}(;i-k) = c_{-k}(;i-k) + a_{i-k}(k;)b_{-i}$$

for $i = k+1, ..., n-1$,

$$c_{-k} = c_{-k} + a_{-i}(k-i;)b_{i-k}(;i)$$

for $i = 0, 1, ..., k$.

To estimate the total number of miner cycles required to perform this algorithm on the STAR-100, we note that

$$\begin{aligned} T_{\text{total}} = {} & 156 \text{ [number of vector multiply operations]} \\ & + 94 \text{ [number of vector addition operations]} \\ & + 1 \text{ [total number of multiplies actually performed]} \\ & + \tfrac{1}{2} \text{ [total number of additions actually performed]}. \end{aligned} \quad (2)$$

A little calculation easily yields $T_{\text{total}} =$

$250[3n^2 - 3n + 1] + \frac{3}{2}n^3$. The number of vector additions could be reduced slightly by properly initializing the vectors $c_{-,k}$ to something other than 0. In any case though, we see that for large values of n, the new diagonal algorithm and the conventional algorithm are essentially equivalent. They differ only in start-up time. For a full matrix, the start-up costs are about three times higher for the diagonal algorithm. Also, the diagonal algorithm will involve somewhat more overhead due to the amount of vector offsetting and looping control which is required. It is worth mentioning that this new algorithm can be easily implemented on other vector/parallel processors such as the TI-ASC and ILLIAC-IV machines. However, on the ILLIAC IV there are other methods which are more efficient for full matrices and we refer the reader to the unpublished report of Stevens [1].

5. Banded matrix algorithm

The real usefulness of matrix multiplication by diagonals occurs when the matrices A and B have banded structures. Suppose $A = (a_{-L_1}, ..., a_0, ..., a_{U_1})$ is a banded matrix of order n, with L_1 lower and U_1 upper diagonals and B (of order n) is similarly given by $B = (b_{-L_2}, ..., b_0, ..., b_{U_2})$. Then, the matrix $C = AB$ can be described by $C = (c_{-L_3}, ..., c_0, ..., c_{U_3})$ where $L_3 = \min(n - 1, L_1 + L_2)$ and $U_3 = \min(n - 1, U_1 + U_2)$.

The matrix multiplication algorithm of Theorem 1 immediately generalizes to the band matrix case and we have the following theorem.

Theorem 2. If A and B are banded matrices as described above, then $AB = C = (c_{-L_3}, ..., c_0, ..., c_{U_3})$ may be computed by the following diagonal algorithm: let $c_k \equiv 0$ for all k, then,

i) For $k = 0, 1, ..., U_3$, compute c_k by

$$c_k(i - k;) = c_k(i - k;) + a_{k-1}(;k)b_i$$

$$\text{for } i = k + 1, ..., \min(U_2, k + L_1),$$

$$c_k(;i - k) = c_k(;i - k) + a_i b_{k-i}(k;)$$

$$\text{for } i = k + 1, ..., \min(U_1, k + L_2),$$

$$c_k = c_k + a_i(;k - i)b_{k-i}(i;)$$

$$\text{for } i = \max(0, k - U_2), ..., \min(k, U_1),$$

ii) For $k = 1, ..., L_3$, compute c_{-k} by

$$c_{-k}(i - k;) = c_{-k}(i - k;) + a_{-i}b_{i-k}(;k)$$

$$\text{for } i = k + 1, ..., \min(L_1, k + U_2),$$

$$c_{-k}(;i - k) = c_{-k}(;i - k) + a_{i-k}(k;)b_{-i}$$

$$\text{for } i = k + 1, ..., \min(L_2, k + U_1),$$

$$c_{-k} = c_{-k} + a_{-i}(k - i;)b_{i-k}(;i)$$

$$\text{for } i = \max(0, k - L_2), ..., \min(k, L_1).$$

To illustrate the fact that the diagonal algorithm is much more efficient for narrow banded matrices than the conventional algorithm, we will consider the multiplication of two matrices A and B of order n and with $L_1 = L_2 = U_1 = U_2 = p$. We will assume that p is small relative to n so that $C = AB$ is also a banded matrix with $L_3 = U_3 = 2p$.

To compute the typical colum of C using the conventional algorithm would require essentially $2p + 1$ vector multiplications and additions each of length $2p + 1$. Therefore, since there are n columns to compute, from (2) we have that

$$T_{\text{conv}} \simeq 250[n(2p + 1)] + \tfrac{3}{2}n(2p + 1)^2.$$

The large number of distinct vector operations is clearly due to the large number of vectors needed to describe the matrix when it is stored by columns. In contrast, we note that for the diagonal algorithm, it requires only $2p + 1 - k$ vector multiplications and additions to compute c_k. Since the total number of scalar operations actually performed is the same for either algorithm we have from (2) that $T_{\text{diag}} = 250(2p + 1)^2 + \frac{3}{2}n(2p + 1)^2$. For a particular case, consider the estimated times (in minor cycles) for the multiplications of two tridiagonal matrices where $n = 1000$ and $p = 1$. Clearly we have

$$T_{\text{conv}}/T_{\text{diag}} = 763500/15750 \cong 48.5,$$

which shows that multiplication by diagonals is much more efficient than the conventional algorithm.

6. Matrix-vector multiplication

A question which immediately comes to mind is: if a matrix is stored by diagonals, can a matrix-vector multiplication be performed conveniently, when the vector is stored as a column? The answer is given by the following theorem.

Theorem 3. If $A = (a_{-L}, \ldots, a_0, \ldots, a_U)$ is a banded $n \times n$ matrix stored by diagonals and x is a column vector of length n, then $b = Ax$ may be computed by the following "Fortran-like" algorithm: let $b = 0$, then

i) For $k = 0, 1, \ldots, U$ compute

$$b(:k) = b(:k) + a_k x(:k),$$

ii) For $k = 1, \ldots, L$ compute

$$b(k:) = b(k:) + a_{-k} x(:k).$$

Again, the only difference in the estimated times for the execution of this algorithm and that for the conventional algorithm lies in the total start-up times. The diagonal algorithm would require $U + L + 1$ vector multiplications and the same number of additions, whereas the conventional algorithm would require n of each type.

For narrow banded matrices where $U + L + 1 \ll n$, the diagonal matrix-times-vector algorithm requires much less total vector operation *start-up* time. For full matrices, the diagonal matrix-times-vector algorithm requires about twice as much *start-up* time, as does the conventional algorithm.

Thus we see that for matrices *of sufficient size*, the diagonal algorithms are slightly less efficient than the conventional algorithm for full matrices (differing only in total start-up time), and much more efficient for narrow banded matrices.

Acknowledgment

The authors are grateful to the referee for his comments and for bringing the Stevens reference to our attention.

Reference

[1] J.E. Stevens, Jr., Matrix Multiplication Algorithm for ILLIAC IV, Dept. of Computer Science, University of Illinois at Urbana, ILLIAC IV Document No. 231, DCS File No. 855, August 26, 1970.